(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 998 076 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
03.05.2000 Bulletin 2000/18

(51) Int Cl.[7]: **H04L 12/26**, H04L 12/24

(21) Application number: **99308205.6**

(22) Date of filing: **18.10.1999**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **30.10.1998 US 184101**

(71) Applicant: **LUCENT TECHNOLOGIES INC.**
Murray Hill, New Jersey 07974-0636 (US)

(72) Inventors:
- **Foley, Michael P.**
  Elmwood Park, Illinois 60707 (US)
- **Vangsness, Kurt A.**
  Aurora, Illinois 60506 (US)

(74) Representative:
Buckley, Christopher Simon Thirsk et al
Lucent Technologies (UK) Ltd,
5 Mornington Road
Woodford Green, Essex IG8 0TU (GB)

(54) Method for controlling a network element from a remote workstation

(57) A method is provided for controlling a network element from a client at a remote work station connectable to the network, the network, element is registered for attributes to be tracked, and attributes associated with the network element are polled only if the client requests the monitoring of the network element.

EP 0 998 076 A1

Printed by Jouve, 75001 PARIS (FR)

Description

Background Of The Invention

[0001] This invention relates, generally, to a method for controlling a network element and, more particularly, to a method for remotely controlling the network by communications through the network.

[0002] Network management systems in which network elements, or management agents, are remotely controlled from a remote, management work station by means of communications between the management work station and the managed network elements sent through the network are known. Such known network management systems employ a special communication protocol for communications between the remote work station running a management program and an element management server that contains a management information base that defines the interface between the work station and the network elements.

[0003] Known systems such as (Hewlett Packard HP-OV NNM or DM, Sun Microsystems Solstice) present an interface where the client application must poll the network element when status is needed. In these systems, the polling may not be coordinated and is replicated for each client, if each client is interested in the same attributes. Also, each of the clients receive the full results for each polling cycle (even if there was no change from the last cycle), increasing the bandwidth used to communicate between the client application and the network element, as well as creating additional processing overhead due to the replicated polling at the network element.

Summary Of The Invention

[0004] A method is provided for controlling a network element from a remote work station connectable to the network. The method provides for registering the network element for attributes to be tracked, and polling for attributes associated with the network element only if the client requests the monitoring of the network element. Changes in attributes are reported when the client requests notification of changes in attributes. For attributes polled for a plurality of clients, changes in the attributes to one of the plurality of clients requesting notification of changes in the attributes are reported.

[0005] The method further provides for polling once for a plurality of clients that registers for the same attributes and reporting asynchronously changes in the attributes to a plurality of clients.

[0006] Another aspect of the invention provides for running an object oriented program at the remote work station to control an object associated with the controllable network element, translating interface operations generated by the work station during the running of the object oriented program to corresponding translated interface operations in an object oriented language associated with the object being controlled, and connecting the corresponding translated interface operations through the network to an object server to control the object associated with the network element in accordance with the translated interface operations.

[0007] These and other features and advantages of the present invention will become apparent from the following detailed description, the accompanying drawings and the appended claims.

Brief Description Of The Drawings

[0008] The foregoing advantageous features will be described in detail and other advantageous features of the invention will be made apparent from the detailed description of the preferred embodiment of the invention that is given with reference to the several figures of the drawings, in which:

Fig. 1 is a functional block diagram of the preferred embodiment of a management system using the preferred network element control method of the present invention;

Fig 2 is a functional block diagram of the preferred embodiment of the translating interface shown as a single functional block in Fig. 1;

Fig. 3 is a functional block diagram illustrating the interface with the controlled network element that is visible to object oriented client management application at the work station of Fig.1;

Fig. 4 is a table of a plurality of service objects that interact with the client management application run at the work station of Fig.1;

Fig. 5 is a table of a plurality of call back functions performed at the translating interface of Fig.2;

Fig. 6 is a table of the different fundamental data types capable of being translated by the translating interface of

Fig.2 in accordance with the invention.

Fig. 7 is a block diagram showing the relationship between client application-specific service object, and the internal service representative of managed object instances;

Fig. 8 is a table summarizing filter criteria that is valid for each event category; and

Fig. 9 is a table defining specific exceptions with an EMAPI exception code containing one of the listed values.

Detailed Description

[0009] This invention provides an application programming interface (API) and protocol that provides for efficient communication between a distributed client application and an element management server independent of the communication protocol to the network element. The Element Management Application Programming Interface (EMAPI) provides the following benefits over known management system. The invention has application in the management of a telecommunication network element. For more information regarding such a management of a telecommunication network element refer to commonly owned U. S. patent application serial number 09/088,463, entitled "Method for Computer Internet Remote Management of a Telecommunication Network Element" by William E. Barker, Lisa M. Connelly, Marvin A. Eggert, Michael P. Foley, Kenneth R. Macfarlane, Philip M. Parsons, Girish Rai, Jerome E. Rog, and Kurt A. Vangsness, filed on May 31, 1998, the disclosure of which is hereby incorporated by reference.

- Efficient use over low bandwidth connections. Client applications register for network element information they wish to track and after an initial set of data only receive incremental updates (deltas) when there are changes.

- Centralized polling of attributes. Attributes are only polled if a client exists that has registered to monitor the attribute. If multiple clients register for the same attribute(s), the polling is not repeated for the clients-only a single polling cycle is performed.

[0010] The invention is used in an operations, administration and maintenance system 20 as shown in Fig. 1. The system 20 includes a PC or workstation 22, an element management server (EMS) 24, an interface in accordance with the invention 26, located between the workstation 22 and an object server 25. An application processor 28 is connected to the element management server 24.

[0011] The workstation 22 includes a web browser 30 which is the interface to the client and is a host for JAVA applets 32 and web browser HTML 35 which is a hypertext markup language.

[0012] The system 20 operates on a cluster computing environment, and leverages off-the-shelf technology to enable additional customers visible features, while extending to subsequent releases and other projects, with minimal increased cost. System 20 is provided through the web browser interface and a SNMP based element management platform.

[0013] A client executes applications via web pages at the workstation 22. The client makes requests for various views of the network status by making selections through the web browser 30. The web browser requests pages from the web server 28 which transmits HTML pages that contain instructions to load and run the appropriate JAVA applet 32. One the applet starts, it communicates with the object server 25 through the interface 26 to perform initialization and to request initial configuration and status information that is appropriate for the current requested view. The JAVA applet 32 then registers with the object server 25 for subsequent notifications of changes to configuration and status that it requires to keep the view up to date. The client may perform commands to request various maintenance operations on the network element 28. These commands are converted into appropriate requests through the interface 26 and perform operations on the object server 25. The commands are then translated into SNMP and are transmitted to the network element 28 through the SNMP library 33. Acknowledgements and command responses from the network element 28 are transmitted through the SNMP library 33, are converted to events by the object server 25 and transmitted to originating JAVA applet 32 through the use of callbacks defined by the interface 26.

[0014] In one embodiment of the invention, as shown in Fig. 3, client applications in JAVA applets 32 include an active alarm list browser, a system alarm survey and a network element detailed status display. Client applications communication with the web server 28 via the interface 26 in accordance with the invention, to the element manager through a distributed object request architecture such as CORBA. The interface 26 provides a constant interface to all managed objects in the network, and hides the implementation details associated with the element manager platform.

[0015] The interface 26 (EMAPI) is the definition of objects, attributes and operations that comprise the protocol used between client applications and the server to manage network elements. The EMAPI uses the industry standard CORBA to provide distribution of the objects and their operations and to allow for the implementation of the client and server

to be in different programming languages and on different computer architectures.

[0016] The client interface to the server and the managed object attributes is described in the interface 26 and managed object notation provides a consistent model of all managed objects in the network, hiding the implementation details associated with the element manager platform from client applications, thus clients do not need to know the underlying protocol to the network elements. Managed objects specific logic is encapsulated within the managed object instead of scattered throughout various applications thus simplifying client application development.

[0017] Each physical, selected non-physical and logical component in the network is modeled as a managed object, which the Server makes visible to distributed client applications through the facilities of the Common Object Request Broker Architecture (CORBA). EM clients need only be concerned about the attributes and operations defined for each application managed object, and not the details of network-level protocol and the server infrastructure required to support object services.

EMAPI Object Definition

[0018] Fig. 3 illustrates all of the interfaces visible to client applications which does not depict process or processor boundaries, which are made transparent by the client and server object request brokers (ORBs). Application services are provided through object interfaces formally defined in the CORBA Interface Definition Language (IDL). The IDL specification of the interfaces described in this document is provided in the Appendix A.

[0019] The service objects resident on the server with which client applications will interact are shown in Fig. 4.

[0020] Client applications which register for real-time status updates or notification of events, alarms or configuration changes must provide a reference to a local callback object which the server will use to propagate information asynchronously. The callback interfaces defined in the interface 26 are shown in Fig. 5. Classes which implement these interfaces must be defined and instantiated in client code.

Data Representation

[0021] There are several fundamental data types defined in the interface 26, which fall into one of the two categories shown in Fig. 6.

Session Management

[0022] Each EM client session is logically associated with a unique login-host combination. Multiple client applications may be associated with the same session, though only one need be registered for the session to be considered active. Session and application identifiers are assigned by the User Session Manager to track resources used by the client, and in future releases, to correlate client access permissions with operation requests. Applications may or may not cross process boundaries. For example, multiple instances of the EMS Command Line Interface (CLI) application registered with the same login and host name will share the same session id, but each process is associated with a different application id. In the EMS Graphical User Interface, all application frames execute in the same process space (albeit in different Threads), yet each frame is associated with a distinct application id. Note that each client application is required to independently register a periodic heartbeat to validate for the Server that its associated resources are still needed.

[0023] The UserSession service object provides the following interfaces:

- startApplication
  This method must be invoked for each client application initialization.
- stopApplication
  A client invokes this method to notify the Server that a target application is terminating, and its associated resources should be released.
- stop
  This method may be used to deregister all applications associated with the same session identifier.
- heartbeat

[0024] This method must be invoked at least every UserSession::HeartbeatPeriod seconds to avoid a timeout condition which, when detected by a Server audit, will result in the release of all resources utilized by an application.

[0025] Refer to the description of interface UserSession in the attachment for additional details.

Managed Objects

[0026] A managed object (MO) is an abstract representation of a physical or logical resource which may be managed by the EMS, such as a network element, maintenance unit or data link. The EM Server will implement one application-specific service object for each type of physical or logical resource to be managed. Each of these service objects defines a set of attributes which identify managed object properties, as well as the operations which may be performed on a specified managed object instance. (The decision to provide access to instance information through a single "service object" stems from the fact that current ORB implementations become unstable when managing very large numbers of remote references.) Fig. 7 depicts the relationship between Client, application-specific service object, and the internal Server representation of managed object instances.

[0027] Each managed object service class is uniquely identified by a ClassCode. Each managed object instance is uniquely identified by an InstId. Any object instance in the system may be uniquely referenced by a managed object identifier (Oid), which is the combination of ClassCode and InstId.

[0028] Managed object status information is reported by a service object as a sequence of attribute code-value pairs. Each attribute value is defined as a union of all of the interface 26 fundamental data types described in Fig. 6.

[0029] Configuration information is reported as a sequence of ConfigData structures, which are defined to contain:

- network element instance id
- managed object instance id
- a managed object key list reported as a sequence of attribute-value pairs-- when length is greater than 0, the key list specifies the associated logical identifiers (LogicalIds)

[0030] Each managed object service class must implement the MO interface, which defines the following configuration and status services:

- viewConfig
  A client uses this method to obtain the current EMS view of the managed object configuration for a specified network element instance. Note that the reserved instance identifier AnyInstance may be used to obtain configuration information for all network elements.
- notifyConfig
  A client may also register for an initial view of managed object configuration information and notification of subsequent changes via callback. The initial view is returned with a notification type CONFIG_INIT. Subsequent changes are reported with type CONFIG_CREATE or CONFIG_DELETE.
- cancelNotify
  A client uses this method to cancel registration for managed object configuration notifications associated with a specified client application.
- getPersistent
  A client may use this method to retrieve the set of attribute codes (SeqAttrCode) identifying all "persistent" data maintained by this service object. Values for persistent attributes of each managed object instance are stored and kept current irrespective of any client requests.
- getAttrSpec
  A client may use this method to retrieve the name and codes of all attributes defined for a target service class (currently used for debugging only).
- getKeySpec
  A client may use this method to retrieve the set of codes (SeqAttrCode) identifying the attribute(s) which represents the logical identifier(s) of any instance of the target class.
- viewStatus
  A client may invoke this method to obtain the EMS view of the current values for a specified set of persistent attributes for a specified managed object instance.

- getStatus
  A client may use this method to register for a snapshot of current status information. This interface differs from the previous one in that the requested attribute list may specify any managed object attribute codes--not just those associated with persistent data, and the information is returned via client status callback (StatusCB).

- startUpdate
  A client may also register for an initial view and notification of any updates to a list of selected attributes for a specified managed object instance. In this case, an initial view is reported via client callback with a notification type STATUS_INIT. Subsequent changes are reported with type STATUS_CHANGE. Note that managed object instance deletions are reported only through configuration change notification to avoid a potential flurry of client status callbacks when a network element is unequipped.

- stopUpdate
  A client uses this method to cancel registration for managed object status updates associated with a specified client application.

- getInst
  A client may use this method to obtain a managed object instance identifier for a specified network element instance id and managed object key list.

[0031] Note that each method requires a client session application identifier (SessionAppId) to validate user access. In the case of configuration or status change notification registration, this identifier is also used to keep track of the additional server resources utilized while the client application is active.

[0032] Refer to the description of interfaces MO, ConfigCB & StatusCB in the attachment for additional details.

Network Element Level Managed Objects

[0033] Each network-element level managed object must also implement the NEMO interface which defines additional network-element level configuration services:

viewNEconfig

[0034] A client may invoke this method to obtain the current EMS view of the network element configuration.

- notifyNEconfig
  A client may also register for an initial view of network element-level managed object configuration information and notification of subsequent changes via callback. The initial view is returned with a notification type CONFIG_INIT. Subsequent changes are reported with type CONFIG_CREATE or CONFIG_DELETE.

- cancelNEnotify
  A client application should use this method to cancel registration for network element managed object configuration updates.

- getNEinst
  A client may invoke this method to retrieve the NEMO instance identifier of the network element associated with a specified logical id.

- getLogicalId
  A client may invoke this method to retrieve the logical identifier of the network element associated with a specified NEMO instance id.

- getContainment
  A client may invoke this method to obtain a sequence of containment information for the target NEMO, where each entry in the sequence contains the name, class code and CORBA reference to a contained service class object.

[0035] Note that each method requires a client session application identifier to validate user access. In the case of configuration change notification registration, this identifier is also used to keep track of the additional server resources

utilized while the client application is active.

[0036] Refer to the description of interfaces NEMO & NEconfigCB in the attachment for additional details.

Descriptive Entity Objects

[0037] Application objects of this type are defined to provide type and attribute information for abstract entities, such as data communicated between the EMS and network elements which are not part of a managed object description (e.g. SNMP trap definitions and command groups). Descriptive entity objects provide no implementation-they are defined in application-specific IDL and known by client applications at compile time.

Event Distributor

[0038] An event is reported as a combination of the following:

1. A header, which contains information of most general interest:

- Time of the event
- Event category defined to be one of the following:
  - * Alarm Set
  - * Alarm Clear
  - * Command Acknowledgment
  - * Command Response
  - * Configuration Change
  - * Informational Message
  - * Initialization
  - * State Change
    - Network element object identifier
    - Network element alarm level-meaningful only for alarm set
    - Maintenance unit object identifier (if applicable)
    - Maintenance unit alarm level-meaningful only for alarm set
    - A command identifier (CmdId) defined as a user session id & command sequence number-meaningful only for command acknowledgment & response

1. Event data defined as a sequence of structures which contain:

- A ClassCode of a managed object, network element or descriptive entity
- A sequence of attribute code-value pairs

[0039] Client applications may request a copy of the event stream, as processed by the event distributor, filtered on information specified in the event header. Filter wildcards are implemented with "out-of-band" values:

- Any Category

- Any Class
- Any Instance
- Any Alarm
- Any Cmd

[0040] The table in Fig. 8 summarizes which filter criteria are valid for each event category:

[0041] The event distributor processes filters by examining the specified category and AND'ing together valid criteria. Clients may simulate OR operations by registering multiple filters.

[0042] The EvtDist service object implements the following client interfaces:

- RegisterFilter
  A client uses this method to register an event filter. A filter identifier is returned.
- CancelFilter
  A client invokes this method to remove a specified event filter, using the filter id returned from the associated registration.

[0043] Note that each method requires a client session application identifier to validate user access.

[0044] Refer to the description of interfaces EvtDist & EventCB in the attachment for additional details.

Alarm Manager

[0045] Alarm information is reported as a sequence of AlarmData structures which contain:

- The ClassCode of a managed object which defines a network-element specific alarm record.
  Note that in the first release of the EMS, only one network element active alarm table is defined (ApActiveAlarms).
- A sequence of alarm records, each of which contains an alarm instance identifier and sequence of attribute code-value pairs.
  Client applications may request a copy of all active alarms filtered on any combination of the following:
  - Network element
  - Maintenance unit
  - Alarm level

[0046] Similar to the interfaces provided by the event distributor, out-of-band values may be used to represent wildcards.

[0047] Since managed object instance information may not be available at the time an alarm is reported, the actual alarm filter criteria are specified in terms of logical identifiers. Logical ids are integer values which represent the logical numbers of devices and interfaces (e.g. AP 4). The correlation between logical ids and managed object instance identifiers is provided in the configuration information made available by each managed object service object, and through the utility method getInst. Refer to the section on Managed Objects for additional details.

[0048] The following AlarmManager client interfaces are written specifically for the Active Alarm List application:

- RequestAlarms
  A client invokes this method to register a filter for active alarms.
- ChangeFilter
  A client may invoke this method to change filter criteria.
- RefreshAlarms
  A client may invoke this method to refresh the active alarm list.

- CancelAlarms
  A client should invoke this method to de-register a filter.

[0049] All operations except for de-registration return all active alarms filtered on the specified criteria. Also, each of these methods requires a valid client session application identifier to validate user access, and to keep track of the additional server resources which may be utilized while each client is active.

[0050] The following AlarmManager interface may be used by any client application (e.g. CLI):

- opAlarm
  Through client implementations of event callbacks used to process command acknowledgements and responses (the same EventCB reference may be used in both cases), this method returns either a list of all active alarms in the system or just those associated with a target network element.

[0051] Refer to the description of interfaces AlarmManager, AlarmCB & EventCB in the attachment for additional details.

Exceptions

[0052] Exceptions are used for consistent and structured error handling in both the EM Server and Client.

[0053] The CORBA specification defines many system exceptions: •

- BAD_PARAM
- INV_OBJREF
- NO_PERMISSION
- BAD_OPERATION
- OBJ_ADAPTER
- ♦♦♦

[0054] Refer to "The Common Object Request Broker: Architecture and Specification" for an exhaustive list of mnemonics and the associated exception descriptions.

[0055] Vendor-specific object request broker exceptions are also defined (using the Minor identifier of the SystemException):

- NO_IT_DAEMON_PORT
- LICENCE_EXPIRED
- ♦♦♦

[0056] Currently, the EMS uses Iona's Orbix product. Refer to the "Orbix 2.3c Reference Guide" for an exhaustive list of mnemonics and the associated exception descriptions.

[0057] In most cases, exceptions will be treated as fatal errors by Client code resulting in application termination.

[0058] An interface 26-specific exception is also defined as an Exception Code containing one of the following values shown in Fig. 9.

APPENDIX A- Emapi.idl

```
#ifndef _EMAPI_IDL
#define _EMAPI_IDL

//
// File: Emapi.idl
//
// Description: CORBA IDL file for the Element Manager API. All client
//      visible interfaces that are common to all applications of the
EMS
//      are described here.
//

// interfaces which are referenced before they are defined
interface ConfigCB;
interface StatusCB;
interface NEconfigCB;
interface EventCB;
interface AlarmCB;

// some of the more commonly used ASN.1 standard types
typedef boolean                  Asn1Boolean;
typedef long                     Asn1Integer;
typedef unsigned long            Asn1UInteger;
typedef sequence<octet>          Asn1Octet;
typedef unsigned long            Asn1Timeticks;
typedef unsigned long            Asn1Gauge;
typedef unsigned long            Asn1Counter;
typedef octet                    Asn1IpAddress[4];
typedef octet                    Asn1Null;
typedef sequence<unsigned long> Asn1Oid;


// logical identifier--one or more attributes of this type may be
defined
// for any managed object to provide a logical identity (e.g.
application
// processor number, RCS number)
typedef Asn1Integer              LogicalId;
typedef sequence<LogicalId>      SeqLogicalId;

// definition of an object identifier
typedef long             ClassCode;
typedef unsigned long    InstId;

struct Oid {
    ClassCode            classId;
    InstId               instId;
};

// reserved class id
const ClassCode          AnyClass        = -1;

// reserved instance id's
const InstId             NullInstance    = 0;
const InstId             AnyInstance     = 1;
const InstId             MaxRsvdInst     = 1;
```

```
// instance id associated with singleton objects (e.g. System)--note
that
// this is the first id normally assigned
const InstId            SingletonInst   = MaxRsvdInst + 1;

// reserved logical id
const LogicalId         AnyLogicalId    = -1;

// application identifier
typedef long            AppId;

// reserved application id's
const AppId             NullAppId       = -1;
const AppId             AnyAppId        = -2;

// application/session identifier
struct SessionAppId {
    InstId              session;
    AppId               app;
};

// maximum number of applications active per single user session
const short             MaxSessionApps  = 10;

// command identifier
typedef long            CmdSeqNo;

// reserved command sequence number
const   CmdSeqNo        AnyCmd          = -1;
const   CmdSeqNo        InvalidCmd      = -1;
struct CmdId {
    InstId              session;
    CmdSeqNo            seqNo;
};

// element manager base application programming interface exception
codes
enum EmapiExceptionCode {
        EM_INVALID_USER,
        EM_UNKNOWN_HOST,
        EM_TOO_MANY_USER_SESSIONS,
        EM_TOO_MANY_APPLICATIONS,
        EM_INVALID_SESSION_ID,
        EM_INVALID_APP_ID,
        EM_INVALID_INST_ID,
        EM_INVALID_NE_ID,
        EM_INVALID_MO_ID,
        EM_INVALID_ATTR_CODE,
        EM_NO_MATCHING_INST,
        EM_INVALID_FILTER,
        EM_INVALID_FILTER_ID,
        EM_NE_ISOLATED,
        EM_INTERNAL_ERROR,
        EM_INVALID_OPERATION,
        EM_ACCESS_DENIED,
        EM_VERSION_MISMATCH,
        EM_LOST_RESOURCE,
        EM_INVALID_KEY,
        EM_INVALID_CATEGORY
```

```
};

// element manager exception
exception EmapiException { EmapiExceptionCode errCode; };

// Access permissions:
//      For now, permission is granted on a network element basis:
//              on any kind of access
//              on status information access
//              on maintenance operation access
//      The network element basis can be:
//              on any network element
//                      Oid: (AnyClass,AnyInstance)
//              on any instance of a particular network element type
//                      Oid: (<class>,AnyInstance)
//              on a specific instance of a particular network element
type
//                      Oid: (<class>,<instance>)

// Kinds of Access (set to be manageable by using a bit map/mask):
typedef AsnlInteger     AccessType;

const   AccessType      AnyAccess = -1;
const   AccessType      NoAccess = 0;
const   AccessType      StatusAccess = 1;
const   AccessType      OperationAccess = 2;
//const AccessType      NextAccess = 4;
//const AccessType      AfterThat = 8
//const AccessType      AfterThat = 16
// ...and so on, in powers of 2

// AccessPermission structure:
//      accessible      TRUE or FALSE for this type of access
//      accessType      type of access
//      oid             network element involved in this type of
access

struct AccessPermission {
        boolean         accessible;
        AccessType      accessType;
        Oid             oid;
};

// Application registration results in the client's receiving a
sequence of
// AccessPermission blocks.

typedef sequence<AccessPermission>      SeqAccessPermission;

struct AccessPermissionList {
        SessionAppId            sessionAppId;
        SeqAccessPermission     seqAccessPermission;
};

// user session
interface UserSession {
        // Null Session
        const InstId            NullSession             = 0;
```

```
        // maximum number of sessions per system
        const short             MaxUserSessions         = 32;

        // heartbeat period in seconds (Applications should use this
value to
        // time their heartbeats with the server.)
        const long              HeartbeatPeriod         = 5;

        // client application registration: must be called once per
        // application.
        //
        // INPUTS:
        //      applicationName: determined by client application
        //      host: name of host on which client is running
        //      user: login id
        //      passwd: user's password
        // RETURNS:
        //      unique id for this session's application
        //      plus a sequence of AccessPermission blocks
        //      showing what is accessible from this application
        // NOTE: The client's responsibility to delete the returned
pointer
        //
        // THROWS:
        //      EM_TOO_MANY_SESSIONS
        //      EM_TOO_MANY_APPLICATIONS
        AccessPermissionList startApplication(
                in string host,
                in string user,
                in string passwd,
                in string applicationName)
                raises(EmapiException);

        // client deregistration -- entire session
        //
        // INPUT: a session application id
        //
        // THROWS:
        //      EM_INVALID_SESSION_ID
        void stop(in InstId sessionId) raises(EmapiException);

        // client deregistration -- single application
        //
        // INPUT: a session application id
        //
        // THROWS:
        //      EM_INVALID_SESSION_ID
        //      EM_INVALID_APPLICATION_ID
        void stopApplication(in SessionAppId id)
raises(EmapiException);

        // client heartbeat
        //
        // INPUT: session application id
        //
        // THROWS:
        //      EM_INVALID_SESSION_ID
        //      EM_INVALID_APP_ID
        //      EM_INTERNAL_ERROR
```

```
        void heartbeat(in SessionAppId id) raises(EmapiException);
};

//
// Managed Object definition. The Managed Object (MO) describes
definitions
// and operations that are common to all application specific managed
// objects in the system.
//
interface MO {
        // All attributes of an MO are identified by integer constant
        // codes called AttrCodes.
        typedef long                    AttrCode;
        const AttrCode                  NullAttrCode    = -1;

        //
        // The following attribute codes are reserved and are used
        // by the MO implementation. Clients should always use the
        // attribute codes found in the application specific managed
        // object definition (e.g. ApModule.idl) and not these.
        //
        const AttrCode                  moInstanceCode  = 0;
        const AttrCode                  neInstanceCode  = 1;
        const AttrCode                  LastReservedCode= 1;

        // clients register for updates by specifying a sequence of
        // attribute codes
        typedef sequence<AttrCode>              SeqAttrCode;

        // The atrribute value is a discriminated union of scalars
        // All basic types currently supported by the EMS are
described
        // here.
        typedef long    AttrType;
        const AttrType  ValInstId               = 1;
        const AttrType  ValAsn1Boolean          = 2;
        const AttrType  ValAsn1Integer          = 3;
        const AttrType  ValAsn1UInteger         = 4;
        const AttrType  ValAsn1Octet            = 5;
        const AttrType  ValAsn1Timeticks        = 6;
        const AttrType  ValAsn1Gauge            = 7;
        const AttrType  ValAsn1Counter          = 8;
        const AttrType  ValAsn1IpAddress        = 9;
        const AttrType  ValAsn1Null             = 10;
        const AttrType  ValAsn1Oid              = 11;
        const AttrType  ValLogicalId            = 12;

        //
        // The following union defines all possible attribute types
        // in the EMS.
        //
        union AttrValue switch(AttrType) {
                case ValInstId:         InstId          instId;
                case ValAsn1Boolean:    Asn1Boolean     asn1Boolean;
                case ValAsn1Integer:    Asn1Integer     asn1Integer;
                case ValAsn1UInteger:   Asn1UInteger    asn1UInteger;
                case ValAsn1Octet:      Asn1Octet       asn1Octet;
                case ValAsn1Timeticks:  Asn1Timeticks   asn1Timeticks;
                case ValAsn1Gauge:      Asn1Gauge       asn1Gauge;
```

```
                case ValAsn1Counter:    Asn1Counter     asn1Counter;
                case ValAsn1IpAddress:  Asn1IpAddress   asn1IpAddress;
                case ValAsn1Null:       Asn1Null        asn1Null;
                case ValAsn1Oid:        Asn1Oid         asn1Oid;
                case ValLogicalId:      LogicalId       logicalId;
        };

        //
        // each set of updates on a managed object is delivered
        // as a sequence of attribute-value pairs (SeqAttrCodeValue).
        //
        struct AttrCodeValue {
                AttrCode        code;
                AttrValue       value;
        };
        typedef sequence<AttrCodeValue> SeqAttrCodeValue;

        //
        // The getAttrInfo method of the MO returns a sequence of
        // information that describes each available attribute.
        //
        struct AttrInfo {
                AttrCode        code;
                AttrType        type;   // note: currently not
populated
                Asn1Octet       name;
        };

        typedef sequence<AttrInfo> SeqAttrInfo;

        //
        // Config update notifications (via the deliverConfig method
in the
        // ConfigCB) can take two forms.
        //      1) One or more managed object instances have been
created.
        //              (CONFIG_CREATE).
        //      2) One or more managed object instances have been
deleted
        //              (CONFIG_DELETE).
        //
        enum ConfigNotifyType {
                CONFIG_CREATE,  // a new MO instance has been created
                CONFIG_DELETE   // an existing MO instance has been
deleted.
        };

        //
        // Status update notifications (via the deliverStatus method
in the
        // StatusCB) can take two forms.
        //      1) An initial update (STATUS_INIT) that is returned
either
        //       as a result of a getStatus or as the initial status
from
        //       a startUpdate request.
        //      2) An inremental update (STATUS_CHANGE) representing a
change
```

```
        //      to one or more of the attributes that the client
registered for.
        //
        enum StatusNotifyType {
                STATUS_INIT,    // represents initial status update of
all
                                // requested attributes (e.g.
startUpdate)
                STATUS_CHANGE   // represents an incremental status
update
                                // (contains only attributes that have
                                // changed).
        };

        //
        // each config update notification will be delivered as a
sequence of:
        //              network element instance id
        //              managed object key list (sequence of attr-
value pairs)
        //              managed object instance id
        struct ConfigData {
                InstId                  neInst;
                SeqAttrCodeValue        keyList;
                InstId                  moInst;
        };

        typedef sequence<ConfigData>    SeqConfigData;

        //
        // viewConfig() - obtain EMS view of the current managed
object
        //      configuration for a specified network element
instance.
        //      The special value AnyInstance may be used to obtain
        //      configuration information for all network elements
known
        //      to the EMS.
        // INPUTS:
        //      sessionAppId - client session/application identifier.
This is
        //                     used to validate client permission to
perform
        //                     the operation.
        //      neInst       - specific NE instance identifier, or
AnyInstance
        //                     to get config for all NE instances.
        // RETURNS:
        //      A sequence of configuration information (sequence
length
        //      is proportional to the number of configured MO
instances).
        //      CALLER MUST DELETE RETURNED MEMORY.
        //
        // THROWS:
        //      EM_INVALID_SESSION_ID
        //      EM_INVALID_APP_ID
        //      EM_INVALID_INST_ID
        //
```

```
        SeqConfigData viewConfig(in SessionAppId    sessionAppId,
                                 in InstId          neInst)
                                 raises(EmapiException);

        //
        // notifyConfig() - Client registration for managed object
        //      configuration information for specified network
element
        //      instance (or the AnyInstance to register for
notifications
        //      on all network element instances). The current
snapshot
        //      of configuration is returned (like viewConfig) and all
        //      subsequent configuration changes result in an
invocation of
        //      the specified callback.
        //
        // INPUTS:
        //      sessionAppId - client session/application identifier.
This is
        //                     used to validate client permission to
perform
        //                     the operation.
        //      neInst       - specific NE instance identifier, or
AnyInstance
        //                     to get config for all NE instances.
        //      callback     - client callback (implements the
ConfigCB
        //                     interface) that will be invoked for
all
        //                     config changes.
        // RETURNS:
        //      A sequence of configuration information (sequence
length
        //      is proportional to the number of configured MO
instances).
        //      CALLER MUST DELETE RETURNED MEMORY.
        //
        // THROWS:
        //      EM_INVALID_SESSION_ID
        //      EM_INVALID_APP_ID
        //      EM_INVALID_INST_ID
        //
        SeqConfigData notifyConfig(in SessionAppId      sessionAppId,
                                   in InstId            neInst,
                                   in ConfigCB          callback)
                                   raises(EmapiException);

        //
        // cancelNotify() - Cancel all requests for configuration
change
        //      notifications associated with the specified client
application.
        //
        // INPUTS:
        //      sessionAppId - client session/application identifier.
This is
        //                     used to validate client permission to
perform
```

```
            //                          the operation.
            // RETURNS:
            //      void
            //
            // THROWS:
            //      EM_INVALID_SESSION_ID
            //      EM_INVALID_APP_ID
            //
            void cancelNotify(in SessionAppId sessionAppId)
        raises(EmapiException);

            //
            // getPersistent() - Obtain the attribute codes for all
        persistent
            //      attributes maintained by this managed object.
        Persistent
            //      attributes are those that the EMS keeps the current
        value
            //      regardless of client registrations. Other attributes
        are
            //      polled for only when a client makes a request or
        registers
            //      for update notifications.
            // INPUTS:
            //      sessionAppId - client session/application identifier.
        This is
            //                          used to validate client permission to
        perform
            //                          the operation.
            // RETURNS:
            //      a sequence of attribute codes representing the
        persistent
            //      attributes.
            //      CALLER MUST DELETE RETURNED MEMORY.
            //
            // THROWS:
            //      EM_INVALID_SESSION_ID
            //      EM_INVALID_APP_ID
            //
            SeqAttrCode getPersistent(in SessionAppId sessionAppId)
                                    raises(EmapiException);

            //
            // getAttrSpec() - Return identification of attributes that
        are
            //      defined for the specific managed object instance. A
        sequence
            //      containing the attribute name (an OCTET representing
        the
            //      ASCII string name) and the associated attribute code
            //      is returned.
            //
            // INPUTS:
            //      sessionAppId - client session/application identifier.
        This is
            //                          used to validate client permission to
        perform
            //                          the operation.
            // RETURNS:
```

```
        //      A sequence of attribute info representing the
                attribute names and codes.
        //      CALLER MUST DELETE RETURNED MEMORY.
        //
        // THROWS:
        //      EM_INVALID_SESSION_ID
        //      EM_INVALID_APP_ID
        //
        SeqAttrInfo getAttrSpec(in SessionAppId sessionAppId)
                                raises(EmapiException);

        //
        // getKeySpec() - Return identification of key attributes. A
sequence
        //      of attribute codes representing the keys is returned.
Note
        //      that the sequence will be in the order defined in the
        //      MOview.
        //
        // INPUTS:
        //      sessionAppId - client session/application identifier.
This is
        //                     used to validate client permission to
perform
        //                     the operation.
        // RETURNS:
        //      A sequence of attribute codes.
        //      CALLER MUST DELETE RETURNED MEMORY.
        //
        // THROWS:
        //      EM_INVALID_SESSION_ID
        //      EM_INVALID_APP_ID
        //
        SeqAttrCode     getKeySpec(in SessionAppId sessionAppId)
                                   raises(EmapiException);

        //
        // viewStatus() - obtain EMS "view" of the values of the          \
specified
        //      persistent attributes.
        // INPUTS:
        //      sessionAppId - client session/application identifier.
This is
        //                     used to validate client permission to
perform
        //                     the operation.
        //      instId       - MO instance. This specifies the MO
whose
        //                     attributes are to be examined.
        //      attrList     - sequence of attribute codes that are to
be
        //                     viewed.
        // RETURNS:
        //      a sequence of attribute code/value pairs representing
the
        //      current view of the attribute values.
        //      CALLER MUST DELETE RETURNED MEMORY.
        //
```

```
        // THROWS:
        //      EM_INVALID_SESSION_ID
        //      EM_INVALID_APP_ID
        //      EM_INVALID_INST_ID
        //      EM_INVALID_MO_ID
        //      EM_INVALID_ATTR_CODE
        //
        SeqAttrCodeValue viewStatus(in SessionAppId     sessionAppId,
                                    in InstId           instId,
                                    in SeqAttrCode      attrList)
                                    raises(EmapiException);

        //
        // getStatus() - request for a snapshot of current status
        //      information. This differs from viewStatus in that
attrList
        //      may specify any managed object attribute codes, and
the
        //      information is returned via client callback. The
callback is
        //      used because the request may require a get request to
the
        //      network element.
        //
        // INPUTS:
        //      sessionAppId - client session/application identifier.
This is
        //                     used to validate client permission to
perform
        //                     the operation.
        //      instId       - MO instance. This specifies the MO
                               attributes are to be examined.
        //      attrList     - sequence of attribute codes that are to
be
        //                     retrieved.
        // RETURNS:
        //      void
        //
        // THROWS:
        //      EM_INVALID_SESSION_ID
        //      EM_INVALID_APP_ID
        //      EM_INVALID_INST_ID
        //      EM_INVALID_MO_ID
        //      EM_INVALID_ATTR_CODE
        //
        void getStatus(in SessionAppId  sessionAppId,
                       in InstId        instId,
                       in SeqAttrCode   attrList,
                       in StatusCB      callback)
                       raises(EmapiException);

        //
        // startUpdate() - client registration for notifications of
        //      any updates to the values of the specified set of
attributes.
        //      An initial snapshot of all requested attributes is
        //      delivered first (type=STATUS_INIT) followed by
notifications
```

```
        //      of only those attributes that have changed
(type=STATUS_CHANGE)
        //      Note that this method may result in polling to the
network
        //      element, but only if the attributes are not persistent
and
        //      no other clients have issued a startUpdate for the
                attributes (only a single poll is used for all
                registered
        //      clients).
        //
        // INPUTS:
        //      sessionAppId - client session/application identifier.
This is
        //                     used to validate client permission to
perform
        //                     the operation.
        //      instId       - MO instance. This specifies the MO
                               whose attributes are to be examined.
        //      attrList     - sequence of attribute codes that are to
be
        //                     monitored.
        // RETURNS:
        //      void
        //
        // THROWS:
        //      EM_INVALID_SESSION_ID
        //      EM_INVALID_APP_ID
        //      EM_INVALID_INST_ID
        //      EM_INVALID_MO_ID
        //      EM_INVALID_ATTR_CODE
        //
        void startUpdate(in SessionAppId  sessionAppId,
                         in InstId        instId,
                         in SeqAttrCode   attrList,
                         in StatusCB      callback)
                         raises(EmapiException);

        //
        // stopUpdate() - client deregistration for status updates.
Cancel
        //      all monitoring activity associated with the specified
client
        //      application. Note that the client may have issued a
number
        //      of startUpdate requests for different instances, but
this
        //      single call to stopUpdate will cancel all of those
requests.
        //
        // INPUTS:
        //      sessionAppId - client session/application identifier.
This is
        //                     used to validate client permission to
perform
        //                     the operation.
        // RETURNS:
        //      void
        //
```

```
        // THROWS:
        //      EM_INVALID_SESSION_ID
        //      EM_INVALID_APP_ID
        //
        void stopUpdate(in SessionAppId sessionAppId)
raises(EmapiException);
        //
        // getInst() - return the instance identifier associated with
the
        //      specified keys.
        //
        // INPUTS:
        //      sessionAppId - client session/application identifier.
This is
        //                     used to validate client permission to
perform
        //                     the operation.
        //      neInst       - the network element instance identifier
of the
        //                     NE that contains the managed object
instance
        //                     specified by the keys.
        // RETURNS:
        //      the InstId of the managed object, or NullInstance if
not found.
        //
        // THROWS:
        //      EM_INVALID_SESSION_ID
        //      EM_INVALID_APP_ID
        //      EM_INVALID_INST_ID
        //
        InstId getInst(in SessionAppId          sessionAppId,
                       in InstId                neInst,
                       in SeqAttrCodeValue      keyValues)
                       raises(EmapiException);
};

//
// client callback for managed object configuration reporting
//
interface ConfigCB {
        oneway void deliverConfig(in ClassCode                  classId,
                                  in MO::ConfigNotifyType       type,
                                  in MO::SeqConfigData          config);
};

//
// client callback for managed object status reporting
//
interface StatusCB {
        oneway void deliverStatus(in Oid                        oid,
                                  in MO::StatusNotifyType       type,
                                  in MO::SeqAttrCodeValue
attrValList);
};

// network element level managed object
interface NEMO : MO {
```

```
        // attribute code reserved for boolean network element
        // isolation indication.
        const AttrCode  IsolatedCode = 1;

        //
        // each network level configuration update notification is
delivered
        // as a sequence of:
        //              network element instance id
        //              network element logical  id
        struct NEconfigData     {
                InstId                  instId;
                LogicalId               logicalId;
        };
        typedef sequence<NEconfigData>  SeqNEconfigData;

        //
        // Sequence of containment information describing all
        // contained managed objects.
        //
        struct  ContainmentInfo {
                ClassCode       classCode;
                MO              moRef;
                Asn1Octet       name;
        };

        typedef sequence<ContainmentInfo> SeqContainmentInfo;

        //
        // viewNEconfig() - obtain EMS view of current network element
        //      configuration
        //
        // INPUTS:
        //      sessionAppId - client session/application identifier.
This is
        //                     used to validate client permission to
perform
        //                     the operation.
        // RETURNS:
        //      Sequence of network element configuration information.
        //      CALLER MUST DELETE RETURNED MEMORY.
        //
        // THROWS:
        //      EM_INVALID_SESSION_ID
        //      EM_INVALID_APP_ID
        //
        SeqNEconfigData viewNEconfig(in SessionAppId sessionAppId)
                                        raises(EmapiException);

        //
        // notifyNEconfig() - client registration for network level
        //      managed object configuration. An initial snapshot of
        //      the network element level configuration is returned.
        //      All subsequent changes to the network element
              configuration are delivered via the specified callback.
        //
        // INPUTS:
        //      sessionAppId - client session/application identifier.
This is
```

```
        //                          used to validate client permission to
perform
        //                          the operation.
        //      callback      - callback object that is to receive
delivery
        //                          of changes to configuration.
        // RETURNS:
        //      Sequence of network element configuration information.
        //      CALLER MUST DELETE RETURNED MEMORY.
        //
        // THROWS:
        //      EM_INVALID_SESSION_ID
        //      EM_INVALID_APP_ID
        //
        SeqNEconfigData notifyNEconfig(in SessionAppId  sessionAppId,
                                       in NEconfigCB    callback)
                                       raises(EmapiException);

        //
        // cancelNEnotify() - cancel request for NE configuration
            change notifications
        //
        // INPUTS:
        //      sessionAppId - client session/application identifier.
This is
        //                          used to validate client permission to
perform
        //                          the operation.
        // RETURNS:
        //      void
        //
        // THROWS:
        //      EM_INVALID_SESSION_ID
        //      EM_INVALID_APP_ID
        //
        void cancelNEnotify(in SessionAppId sessionAppId)
                            raises(EmapiException);

        //
        // getNEInst() - return the network element instance
identifier
        //      associated with the specified NE logical identifier.
        //
        // INPUTS:
        //      sessionAppId - client session/application identifier.
This is
        //                          used to validate client permission to
perform
        //                          the operation.
        //      logicalId     - the logical identifier (integer) of
                                the network element.
        // RETURNS:
        //      the InstId of the network element instance, or
NullInstance
        //      if not found.
        //
        // THROWS:
        //      EM_INVALID_SESSION_ID
        //      EM_INVALID_APP_ID
```

```
        //
        InstId getNEinst(in SessionAppId        sessionAppId,
                         in LogicalId           logicalId)
                         raises(EmapiException);

        //
        // getLogicalId() - return the NE logical identifier
        //      associated with the specified instance identifier.
        //
        // INPUTS:
        //      neInst      - the instance identifier of the network
element.
        //
        // RETURNS:
        //      the logical identifier of the network element
instance,
        //      or 0 if not found.
        //
        AsnlInteger getLogicalId(in InstId neInst);

        //
        // getContainment() - return sequence with containment
information
        //      for this network element. The sequence returned
contains
        //      the name, class code and a pointer to the associated
service
        //      class object.
        //
        // INPUTS:
        //      sessionAppId - client session/application identifier.
This is
        //                     used to validate client permission to
perform
        //                     the operation.
        // RETURNS:
        //      a sequence of containment information describing the
contained
        //      managed objects.
        //      CALLER MUST DELETE RETURNED MEMORY.
        //
        // THROWS:
        //      EM_INVALID_SESSION_ID
        //      EM_INVALID_APP_ID
        //
        SeqContainmentInfo getContainment(in SessionAppId
sessionAppId)
                                          raises(EmapiException);

};

// client callback for network element level managed object
configuration
// reporting
interface NEconfigCB {
        oneway void deliverNEconfig(in ClassCode              classId,
                                    in NEMO::ConfigNotifyType type,
                                    in NEMO::SeqNEconfigData  config);
};
```

```
// typedefs for acknowledgement value & alarm level
typedef AsnlInteger             AckValue;
typedef AsnlInteger             AlarmLevel;

// no alarm filtering conjunctions are permitted, but we do allow a
single
// filter to extract all alarmed notifications by defining an "out-of-
band"
// alarm level
const   AlarmLevel              AnyAlarm        = -1;

// typedef for event data structure--note that instance information
may no
// longer be valid when an event is processed, so key information
should
// always be available in the attribute-value list
struct EventData {
        ClassCode               classCode;
        MO::SeqAttrCodeValue    seqVal;
};
typedef sequence<EventData>     SeqEventData;

// event distributor interface description
interface EvtDist {
        // time of event as reported by EMS
        typedef long            EventTime;

        // categories of events
        enum Category {
                ANY_EVENT,
                ALARM_CLEAR,
                ALARM_SET,
                CMD_ACK,
                CMD_RESP,
                CONFIG_CHANGE,
                INFO_MSG,
                INIT_MSG,
                STATE_CHANGE,
                NUM_CATEGORIES
        };

        // event header (not all members valid for all categories)
        struct Header {
                EventTime       eventTime;
                Category        category;
                Oid             networkElemId;
                AlarmLevel      networkElemAlm;
                Oid             maintUnitId;
                AlarmLevel      maintUnitIdAlm;
                CmdId           cmdId;
        };

        // event filter (not all members valid for all categories)
        struct Filter {
                Category        category;
                Oid             networkElemId;
                AlarmLevel      networkElemAlm;
                Oid             maintUnitId;
```

```
                AlarmLevel      maintUnitIdAlm;
                CmdId           cmdId;
        };

        // cookie for client registration/deregistration
        typedef long            FilterId;

        // client registration for events
        //
        // INPUTS:
        //      id:         SesionAppId associated with the filter
        //      filter:     EvtDist::Filter that specifies filter
                            criteria
        //      callback:   EventCB that defines deliverEvent()
        //                      operation that will be called when an
        //                      incoming event matches the filter
        // RETURNS:
        //      unique FilterId for this filter
        //
        // THROWS:
        //      EM_INVALID_SESSION_ID
        //      EM_INVALID_APP_ID
        //      EM_INVALID_CATEGORY
        FilterId registerFilter(in SessionAppId id,
                                in Filter       filter,
                                in EventCB      callback)
                                raises(EmapiException);

        // client filter deregistration
        //
        // INPUTS:
        //      id:         SesionAppId associated with the filter
        //                      to be cancelled
        //      filterId:   FilterId of the filter to be cancelled
        //
        // THROWS:
        //      EM_INVALID_SESSION_ID
        //      EM_INVALID_APP_ID
        //      EM_INVALID_FILTER_ID
        void cancelFilter(in SessionAppId       id,
                          in FilterId           filterId)
                          raises(EmapiException);
};

// client callback for event notification
interface EventCB {

        // deliverEvent() is called by event distribution
        // software whenever an incoming event matches
        // a registered filter.
        //
        // INPUTS:
        //      header:   EvtDist::Header associated with the
        //                      incoming event
        //      data:     SeqEventData (sequence of event data)
        //                      associated with the incoming
        //                      event
        //
        oneway void deliverEvent(in EvtDist::Header     header,
```

```
                                        in SeqEventData         data);

            // This empty operation is called during event filter
            // auditing to determine whether or not the associated
            // session/application is still active.  If it has
            // terminated, a CORBA::SystemException will be thrown
            // and subsequently caught, indicating that the associated
            // filter should be removed from the filter queue.

            oneway void check();
    };

    // typedefs for alarm data structures
    struct AlarmRecord {
            InstId                  instId;         // unique per alarm
    definition
            MO::SeqAttrCodeValue    attrValList;
    };
    typedef sequence<AlarmRecord>   SeqAlarmRecord;

    struct AlarmData {
            ClassCode               alarmDef;
            SeqAlarmRecord          alarmRecords;
    };
    typedef sequence<AlarmData>     SeqAlarmData;

    // alarm notification type
    enum AlarmNotifyType { ALARM_SET, ALARM_CLEAR };

    // active alarm manager interface description
    interface AlarmManager {
            // active alarm filter--note that logical ids are specified
    since
            // instance information may not be available at the time
    alarms are
            // reported
            struct AlarmFilter {
                    ClassCode       alarmDef;
                    LogicalId       networkElemId;
                    ClassCode       maintUnitClass;
                    SeqLogicalId    maintUnitId;
                    AlarmLevel      alarmLevel;
            };

            // client registration for active alarms--both initial data &
    update
            // notifications (note that only one alarm filter can be
    registered
            // with the alarm manager by each client application)
            SeqAlarmData requestAlarms(in SessionAppId      sessionAppId,
                                       in AlarmFilter       filter,
                                       in AlarmCB           callback)
                                       raises(EmapiException);

            // request to change filter criteria (note that only one alarm
    filter
            // can be registered with the alarm manager by each client
    application)
            SeqAlarmData changeFilter(in SessionAppId       sessionAppId,
```

```
                                        in AlarmFilter        filter)
                                        raises(EmapiException);

        // request to refresh alarm list--returns snapshot of current
alarms
        // (note that only one alarm filter can be registered with the
alarm
        // manager by each client application)
        SeqAlarmData refreshAlarms(in SessionAppId sessionAppId)
                                        raises(EmapiException);

        // client deregistration--cancel alarm set & clear forwarding
to the
        // specified client application (note that only one alarm
filter can
        // be registered with the alarm manager by each client
application)
        void cancelAlarms(in SessionAppId sessionAppId)
raises(EmapiException);
        // request for a list of all alarms in the entire system or
associated
        // with the indicated network element--supports the EMS
version of the
        // "OP:ALARM" technician command
        CmdSeqNo opAlarm(in SessionAppId        sessionAppId,
                         in ClassCode           alarmDef,
                         in LogicalId           networkElemId,
                         in EventCB             ackCB,
                         in EventCB             cmdRespCB)
                         raises(EmapiException);
};

// client callback for active alarm notification
interface AlarmCB {
        oneway void deliverAlarms(in AlarmNotifyType    type,
                                  in AlarmData          data);
};

// active alarms service object interface, derived from managed object
// interface--note that no filtering of active alarms can be specified
interface ActiveAlarms : MO {
        // system client (e.g., Active Alarm Manager) registration for
        // active alarms--both initial data and update notifications
        SeqAlarmRecord getAlarms(in SessionAppId        sessionAppId,
                                 in AlarmCB             callback)
                                 raises(EmapiException);

        // system client (e.g., Active Alarm Manager) deregistration--
        // cancel active alarms update notifications
        void cancelAlarms(in SessionAppId sessionAppId)
raises(EmapiException);
};

#endif // _EMAPI_IDL

      // EOF //
```

APPENDIX B-GLOSSARY

Alarm	The description of an alarmed notification.
Attribute	A property of a managed object (e.g. alarm state).
Attribute Code	An integer value which uniquely identifies an attribute of a given managed object.
Class Code	An integer value which uniquely identifies a managed object class.
Configuration Information	Generic term which has one of two meanings depending on its context: With respect to a managed object class, this term applies to the identification of all instances of the class, either for a specific network element or for all network elements in the system. With respect to a managed object instance, this term may apply to one or more attributes which are associated with database values, such as the primary/alternate role of a duplex component.
CORBA	Common Object Request Broker Architecture
EMAPI	Element Management Application Programming Interface
EMS	Element Management System
Event	The description of a spontaneous occurrence, such as alarm notification, command acknowledgment or configuration change.
Instance Identifier	An integer value which uniquely identifies an instance of a given managed object.
Interface Operation	Generic term for distributed service request. The target method may be defined in the Element Management Application Programming Interface (e.g. status registration) or in an application-specific derivation of a managed object (e.g. command execution).
Logical Identifier	An integer value which represents the logical number of a device or interface (e.g. AP 4). Note that there is no direct correlation between a logical id and instance id.
Managed Object	An abstract representation of a physical or logical resource which may be managed by the EMS (e.g. network element, maintenance unit, data link).
ORB	Object Request Broker
Object Identifier	The combination of managed object class code and instance identifier which

	uniquely identifies any managed object instance.in the system.
Persistent Attribute	Information stored and kept current irrespective of any client request (e.g. maintenance state).
Service Object	Any EM Server object which provides services to client applications.
Session	Each client must establish a session at initialization--for which a unique session identifier is assigned--that will be used to validate access permissions, to correlate client requests and to keep track of Server resources utilized in behalf of any applications associated with the session.
Status Information	Current attribute values for a managed object instance.

Claims

1. In a network having a controllable network element, a method for controlling the network element from a remote work station connectable to the network, comprising the steps of:

   registering the network element for attributes to be tracked; and

   polling for attributes associated with the network element only if the client requests the monitoring of the network element.

2. The method of claim 1 wherein the polling is performed by the server.

3. The method of claim 1 including the step of reporting changes in attributes when the client requests notification of changes in attributes.

4. The method of claim 1, including the steps of

   polling for the attributes for a plurality of clients, and

   reporting changes in the attributes to one of the plurality of clients requesting notification of changes in the attributes.

5. The method of claim 4 wherein the step of polling includes the step of polling once for a plurality of clients that registers for the same attributes.

6. The method of claim 5 including reporting asynchronously changes in the attributes to a plurality of clients.

7. The method of claim 1 wherein the step of polling is performed in a single polling cycle when multiple clients have registered for the same attributes.

8. The method of claim 7 including the steps of

   running an object oriented program at the remote work station to control an object associated with the controllable network element;

   translating interface operations generated by the work station during the running of the object oriented program to corresponding translated interface operations in an object oriented language associated with the object being controlled; and

   connecting the corresponding translated interface operations through the network to an object server to control

the object associated with the network element in accordance with the translated interface operations.

9. The method of claim 8 in which step of translating includes the steps of

automatically determining which of a plurality of different object oriented languages is the object oriented language of the object being controlled, and

generating translated interface operations corresponding to the interface operations from the remote workstation to the object server in accordance with the language of the object being controlled as has been automatically determined.

10. The method of claim 8 in which the object oriented program run at the remote work station is a JAVA program.

11. The method of claim 8 in which the network element is located at a node of the network and the step of translating includes the step of

receiving the interface operations through a communication link of the network at a node separate from the node of the network element,

translating the interface operation through the network communication link into the corresponding translated interface operations by converting the received interface operations into IPC and TCP/IP requests.

12. The method of claim 11 in which the step of connecting includes the step of transmitting the IPC and TCP/IP requests to the object server.

13. The method of claim 12 in which the step of connecting includes the step of generating the IPC and TCP/IP request through a web-based GUI.

14. The method of claim 8 in which the object server responds to the translated object requests by

gathering information concerning the network element, and conveying the information concerning the network node that has been gathered to the remote work station by at least by one of the ways of

dynamically generating a web-page visual display associated with the network element being controlled for interfacing with the remote work station to display the gathered information.

15. The method of claim 14 in which the step of gathering includes the step of gathering network element information concerning at least one of the items of network element information of

a list of all active alarms,

a summary of system alarms, and

a detailed indication of the status of the network element.

16. The method of claim 15 in which the step of gathering includes the step of selectively gathering all of the items of information.

17. The method of claim 8 in which the step of translating includes the step of communication with the object server through a distributed object request architecture to provide a consistent interface to the managed object that hides implementation details associated with the object manager.

18. The method of claim 17 in which the distributed object request architecture is CORBA architecture.

19. The method of claim 8 in which the CORBA architecture functions as an IPC for functions residing on the object server to eliminate the need for platform specific language for the object oriented program at the remote station.

20. The method of claim 8 in which the CORBA architecture functions as an IPC for functions residing on the object

server to provide for distribution of functionality to multiple work station processors.

21. The method of claim 8 in which communication between the network element and the object server is through use of a network management protocol.

22. The method of claim 1 wherein the network management protocol is the simple network management protocol.

23. The method of claim 21 including the step of obtaining system status associated with the network element by polling and auditing pursuant to the simple network management protocol.

24. The method of claim 21 including the step of providing real-time notification of alarm conditions at the network element through the use of network management protocol event manager.

25. The method of claim 21 including the step of providing command and control signals to the network element through use of simple network management protocol set operation.

26. The method of claim 8 in which

the object server is part of an element management server that also includes a web server, and including the step of

displaying command and alarm output information from the network element as a web browser-based display through use of the web server.

27. The method of claim 26 in which

the element management server also includes an executive control processor, and including the step of

sending the command and alarm output information from the network element that is displayed through use of the web server to the executive control processor.

28. The method of claim 12 including the steps of

storing network element information concerning the network at the element management server, and

selectively providing the stored network element information to a plurality of different work stations.

29. The method of claim 8 including the step of sending from the network element event and alarm notifications to the object server through use of a network management protocol.

30. The method of claim 29 including the step of issuing commands from the network element to obtain input information from the work station running the object oriented program to the object server through the use of the network management protocol.

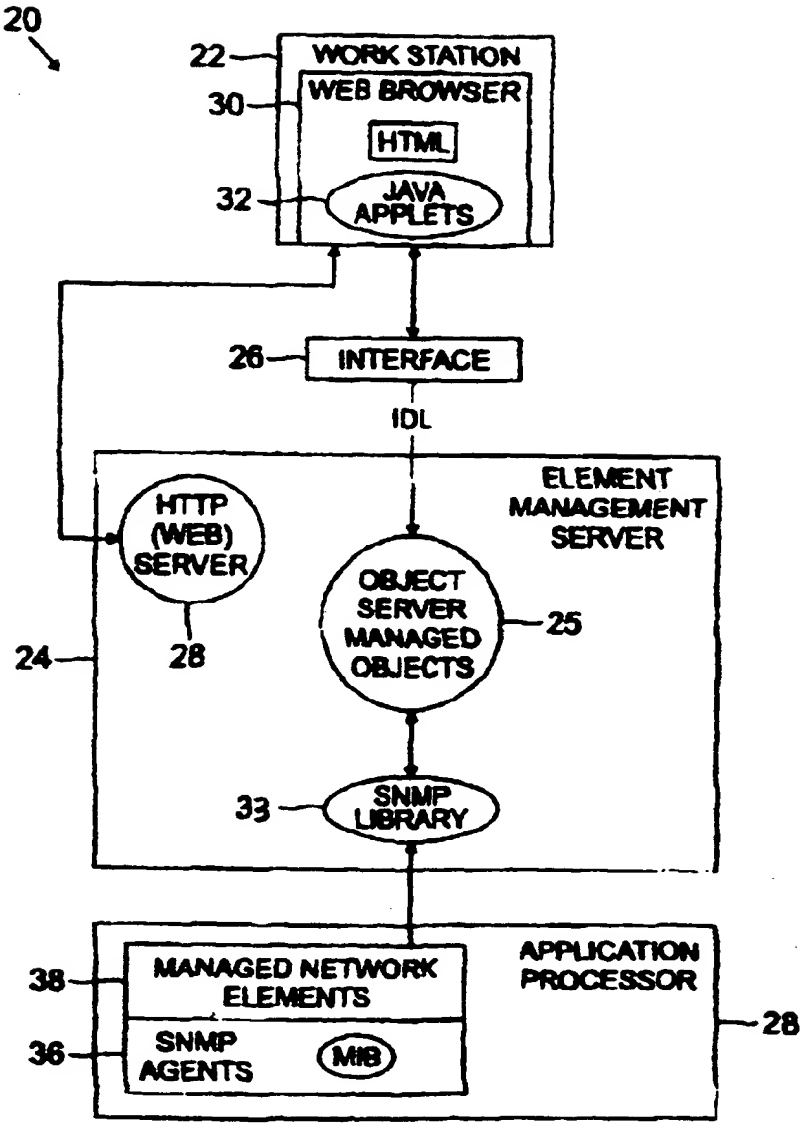


FIG. 1

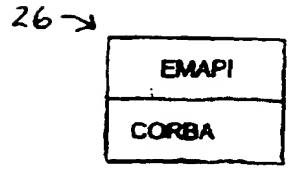


FIG. 2

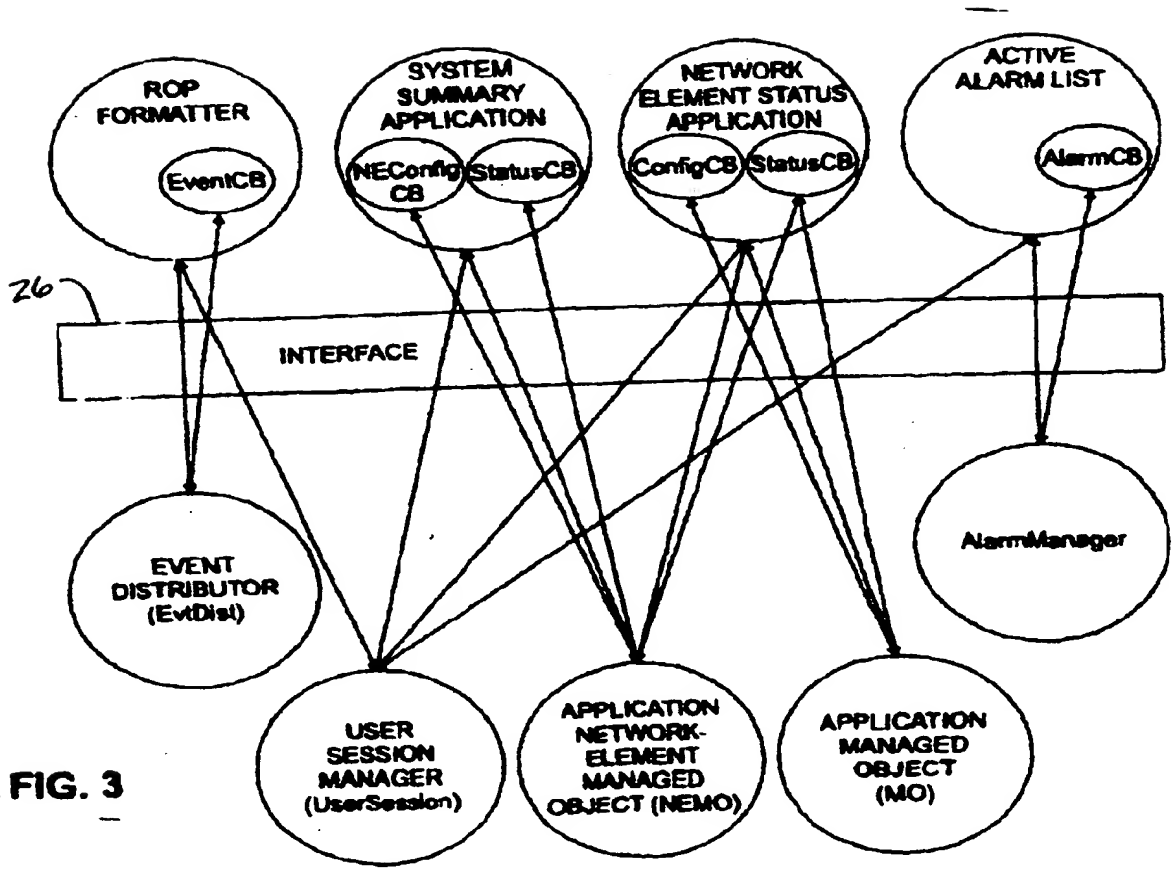


FIG. 3

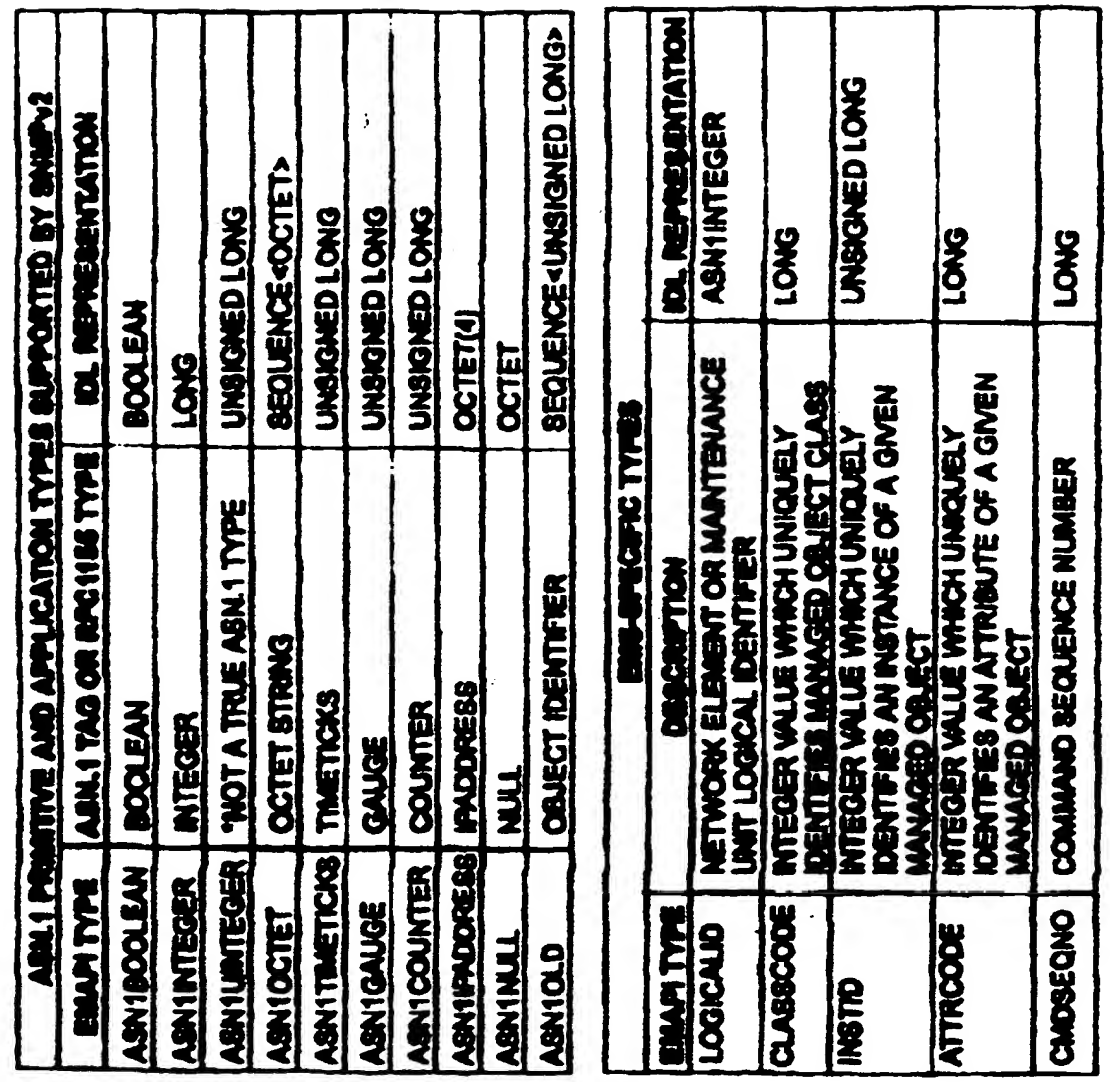
User Session Manager (UserSession)	In its current form, the primary function of the user session manager is to maintain a list of active client sessions and applications. In subsequent releases, this service object will provide user access security on a network-element and operation basis. Refer to the section on Session Management for a discussion of the interfaces provided by UserSession.
Managed Object (MO)	For each physical or logical resource which must be managed by the EMS, an abstract representation will be defined which identifies attributes and operations associated with the resource. Each application-specific managed object implemented on the Server must provide the same Client interfaces for retrieving configuration information, attribute values, and registration for notification of changes. Refer to the section on Managed Objects for further details.
Network-Element Level Managed Object (NEMO)	Each application-specific NEMO implemented on the Server must provide additional interfaces above those provided by the standard managed object to support network-element level configuration queries. Refer to the section on Network Element Level Managed Objects for further details.
Event Distributor (EvtDist)	The event distributor propagates events either received or generated at the Server to clients which register filters specifying event filter criteria. Refer to the section on the Event Distributor for the definition of an event and event filter, and a discussion of the Client interfaces provided by EvtDist.
Alarm Manager (AlarmManager)	The primary utility of the AlarmManager is realized through a single client application called the AlarmList. (Note that there may be more than one instance of the AlarmList application active at any one time). Alarm filters may be registered which filter alarm information based on network element, managed object or alarm level. The AlarmManager returns an initial view of all active alarms matching the specified criteria, and provides notification of changes resulting from subsequent alarm SET or CLEAR events. Refer to the section on the Alarm Manager for the definition of an alarm and alarm filter, and a detailed discussion of the Client interfaces provided by the AlarmManager.

FIG. 4

EP 0 998 076 A1

MANAGED OBJECT CONFIGURATION CALLBACK (ConfigCB)	A ConfigCB OBJECT MUST IMPLEMENT A deliverConfig( ) METHOD FOR NOTIFICATION OF MANAGED OBJECT CONFIGURATION CHANGES. REFER TO THE SECTION ON MANAGED OBJECTS FOR DETAILS ON THE FORMAT OF REPORTED CONFIGURATION DATA.
MANAGED OBJECT STATUS CALLBACK (StatusCB)	A StatusCB OBJECT MUST IMPLEMENT A deliverStatus( ) METHOD FOR NOTIFICATION OF MANAGED OBJECT ATTRIBUTE VALUE CHANGES. REFER TO THE SECTION ON MANAGED OBJECTS FOR DETAILS ON THE FORMAT OF REPORTED STATUS DATA.
NETWORK-ELEMENT LEVEL MANAGED OBJECT CONFIGURATION CALLBACK (NEconfigCB)	AN NEconfigCB OBJECT MUST IMPLEMENT A deliverNEconfig( ) METHOD FOR NOTIFICATION OF NETWORK-ELEMENT LEVEL MANAGED OBJECT CONFIGURATION CHANGES. REFER TO THE SECTION ON NETWORK ELEMENT LEVEL MANAGED OBJECTS FOR DETAILS ON THE FORMAT OF REPORTED NETWORK-ELEMENT LEVEL CONFIGURATION DATA.
EVENT NOTIFICATION CALLBACK (EventCB)	AN EventCB OBJECT MUST IMPLEMENT A deliverEvent( ) METHOD FOR EVENT NOTIFICATION. REFER TO THE SECTION ON THE EVENT DISTRIBUTOR FOR DETAILS ON THE FORMAT OF REPORTED EVENT DATA.
ACTIVE ALARM NOTIFICATION CALLBACK (AlarmCB)	AN AlarmCB OBJECT MUST IMPLEMENT A deliverAlarms( ) METHOD FOR NOTIFICATION OF ACTIVE ALARM CHANGES. REFER TO THE SECTION ON ALARM MANAGER FOR DETAILS ON THE FORMAT OF REPORTED ALARM DATA.

FIG. 5



ASN.1 PRIMITIVE AND APPLICATION TYPES SUPPORTED BY SNMPv2		
SNMPI TYPE	ASN.1 TAG OR RFC1155 TYPE	IDL REPRESENTATION
ASN1BOOLEAN	BOOLEAN	BOOLEAN
ASN1INTEGER	INTEGER	LONG
ASN1UINTEGER	*NOT A TRUE ASN.1 TYPE	UNSIGNED LONG
ASN1OCTET	OCTET STRING	SEQUENCE<OCTET>
ASN1TIMETICKS	TIMETICKS	UNSIGNED LONG
ASN1GAUGE	GAUGE	UNSIGNED LONG
ASN1COUNTER	COUNTER	UNSIGNED LONG
ASN1IPADDRESS	IPADDRESS	OCTET[4]
ASN1NULL	NULL	OCTET
ASN1OLD	OBJECT IDENTIFIER	SEQUENCE<UNSIGNED LONG>

SNMPI-SPECIFIC TYPES		
SNMPI TYPE	DESCRIPTION	IDL REPRESENTATION
LOGICALID	NETWORK ELEMENT OR MAINTENANCE UNIT LOGICAL IDENTIFIER	ASN1INTEGER
CLASSCODE	INTEGER VALUE WHICH UNIQUELY IDENTIFIES MANAGED OBJECT CLASS	LONG
INSTID	INTEGER VALUE WHICH UNIQUELY IDENTIFIES AN INSTANCE OF A GIVEN MANAGED OBJECT	UNSIGNED LONG
ATTRCODE	INTEGER VALUE WHICH UNIQUELY IDENTIFIES AN ATTRIBUTE OF A GIVEN MANAGED OBJECT	LONG
CMDSEQNO	COMMAND SEQUENCE NUMBER	LONG

FIG. 6

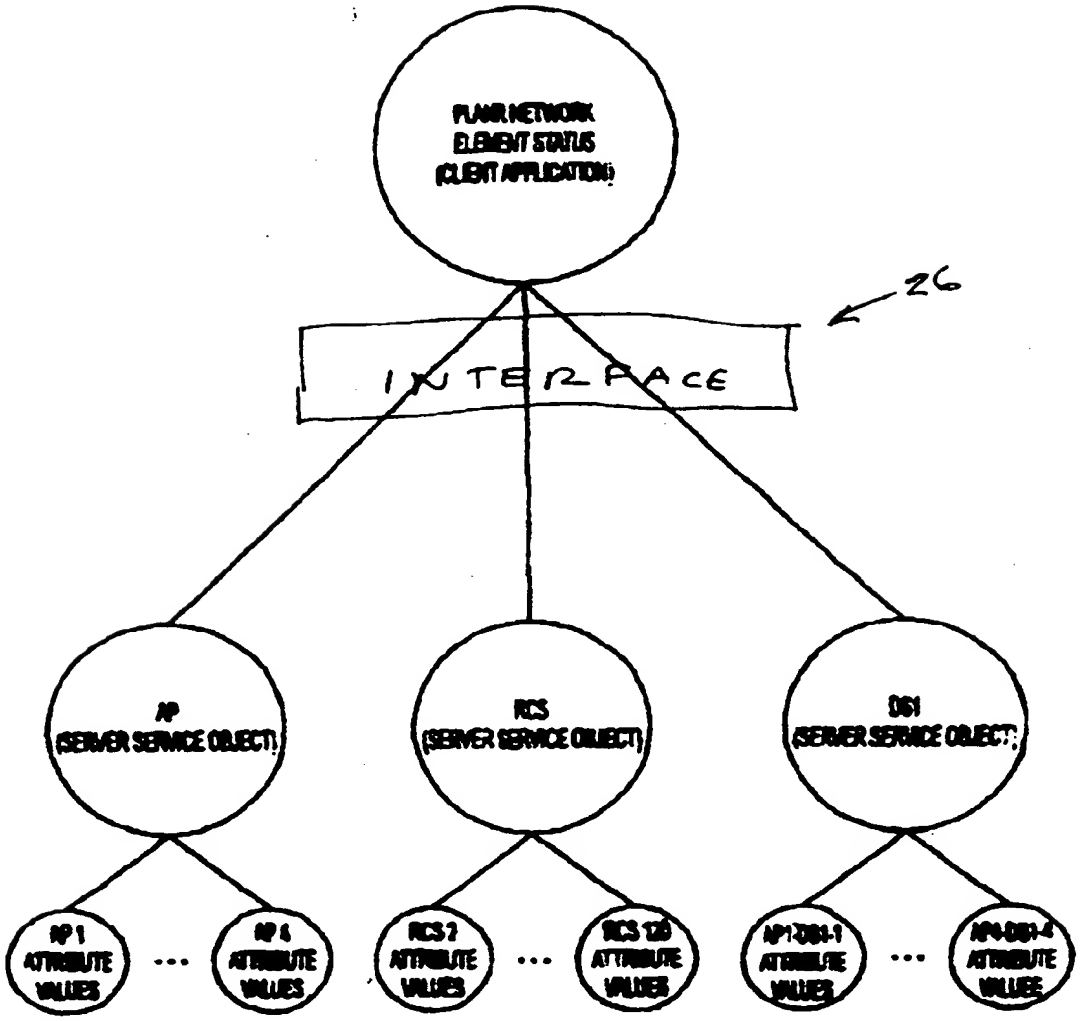
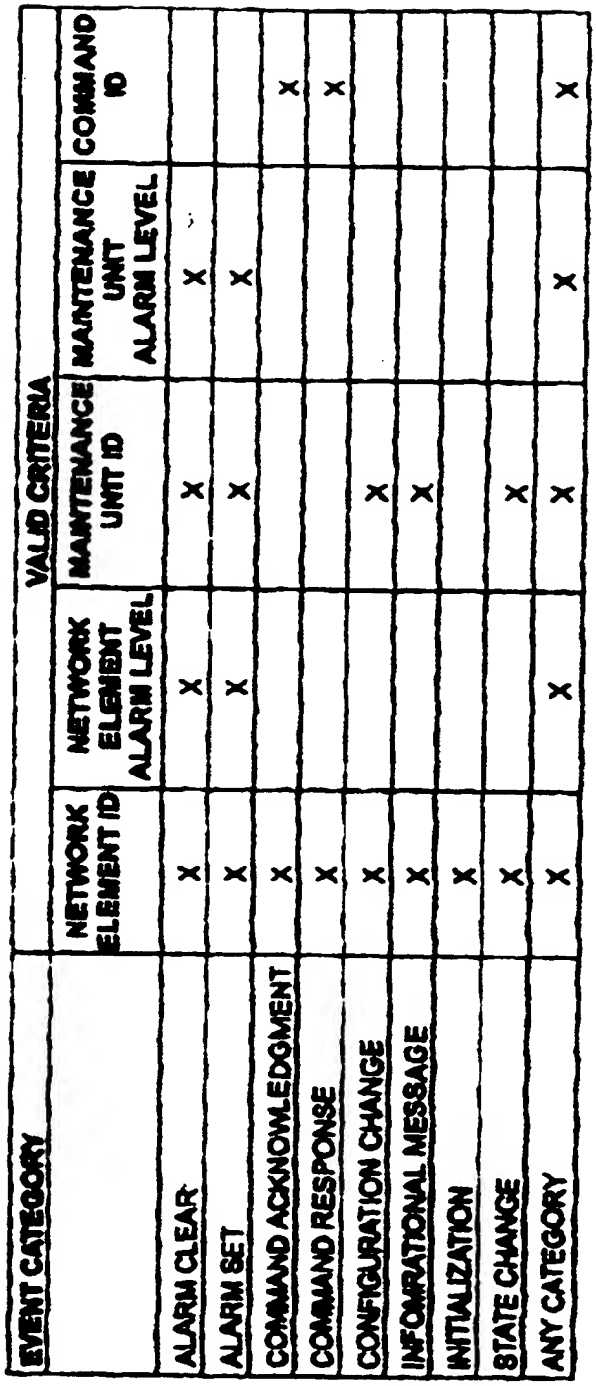


FIG.7



EVENT CATEGORY	VALID CRITERIA				
	NETWORK ELEMENT ID	NETWORK ELEMENT ALARM LEVEL	MAINTENANCE UNIT ID	MAINTENANCE UNIT ALARM LEVEL	COMMAND ID
ALARM CLEAR	X	X	X	X	
ALARM SET	X	X	X	X	
COMMAND ACKNOWLEDGMENT	X				X
COMMAND RESPONSE	X				X
CONFIGURATION CHANGE	X		X		
INFORMATIONAL MESSAGE	X		X		
INITIALIZATION	X				
STATE CHANGE	X		X		
ANY CATEGORY	X	X	X	X	X

FIG. 8

Mnemonic	Meaning
EM_INVALID_USER	The user login identifier is invalid.
EM_UNKNOWN_HOST	The specified host is unknown.
EM_TOO_MANY_USER_SESSIONS	Too many client sessions are already active.
EM_TOO_MANY_APPLICATIONS	Too many client applications are already active for the specified login-host combination.
EM_INVALID_SESSION_ID	The specified client session id is invalid or no longer known.
EM_INVALID_APP_ID	The specified client application id is invalid or no longer known.
EM_INVALID_INST_ID	The specified instance id is invalid (e.g. NullInstance is specified but not accepted in the current application context).
EM_INVALID_NE_ID	The specified network element instance id is invalid or no longer known.
EM_INVALID_MO_ID	The specified managed object instance id is invalid or no longer known.
EM_INVALID_ATTR_CODE	The specified attribute code is not defined for the target managed object.
EM_NO_MATCHING_INST	No managed object instance contains a matching key list.
EM_INVALID_FILTER	The specified filter for either an event or alarm contained one or more invalid criteria.
EM_INVALID_FILTER_ID	The specified filter id for either an event or alarm is invalid or no longer known.
EM_NE_ISOLATED	The specified network element instance is isolated.
EM_INTERNAL_ERROR	The request could not be satisfied because of an EMS Server error.
EM_INVALID_OPERATION	An invalid operation was attempted.
EM_ACCESS_DENIED	Access permission was not granted for the current operation request.
EM_VERSION_MISMATCH	Software version mismatch detected.
EM_LOST_RESOURCE	A critical resource has been lost since the last client application heartbeat (e.g. AlarmManager abnormally terminated).
EM_INVALID_KEY	An invalid key sequence was specified (e.g. wrong number of logical id's specified for a target managed object instance).
EM_INVALID_CATEGORY	An invalid event filter category was specified.

FIG. 9

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 99 30 8205

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
X	US 5 491 796 A (CHEN MICHELE ET AL) 13 February 1996 (1996-02-13)	1,2,4	H04L12/26 H04L12/24
Y	* abstract *	3,7,8, 10,17,18	
A	* figure 1 *	5,6,9, 11-16, 19-30	
	* column 2, line 43-66 * * column 36, line 23 - column 37, line 64 * * claims 1-3,7-9 *		
X	EP 0 831 617 A (DIGITAL EQUIPMENT CORP) 25 March 1998 (1998-03-25)	1,2,22	
A	* abstract *	3-21, 23-30	
	* page 2, line 51 - page 3, line 29 * * figures 1-3 * * claims 1,4,5,7,8 *		
Y	MAGEDANZ T ET AL: "INTELLIGENT AGENTS AN EMERGING TECHNOLOGY FOR NEXT GENERATION TELECOMMUNICATIONS?" PROCEEDINGS OF INFOCOM,US,LOS ALAMITOS, IEEE COMP. SOC. PRESS, vol. CONF. 15, 1996, pages 464-472, XP000621308 ISBN: 0-8186-7293-5	3,7,8, 10,17,18	TECHNICAL FIELDS SEARCHED (Int.Cl.7) H04L
A	* abstract *	1,2,4-6, 9,11-16, 19-30	
	* paragraphs '0002!,'03.4!,'06.1! * * figures 2,4,5 *		
	-/--		

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	7 February 2000	Cichra, M

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 99 30 8205

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
A	KOTSCHENREUTHER J: "BETREIBER BRAUCHEN OFFENE NETZMANAGEMENTSYSTEME" NTZ NACHRICHTENTECHNISCHE ZEITSCHRIFT,DE,VDE VERLAG GMBH. BERLIN, vol. 50, no. 5, 1 January 1997 (1997-01-01), pages 50-52, XP000693391 ISSN: 0027-707X * the whole document *	1-10,17, 18	
A	"DESIGN FOR A SIMPLE NETWORK MANAGEMENT PROTOCOL SUBAGENT FOR INTERNET FIREWALLS" IBM TECHNICAL DISCLOSURE BULLETIN,US,IBM CORP. NEW YORK, vol. 40, no. 3, 1 March 1997 (1997-03-01), pages 63-68, XP000694517 ISSN: 0018-8689	1-4,7,22	

TECHNICAL FIELDS SEARCHED (Int.Cl.7)

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	7 February 2000	Cichra, M

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 99 30 8205

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

07-02-2000

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5491796 A	13-02-1996	AU 5404194 A WO 9410625 A	24-05-1994 11-05-1994
EP 0831617 A	25-03-1998	NONE	

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82